JASPERREPORTS - REPORT SCRIPTLETS

We saw in our previous chapters, data displayed on the report is usually fetched from report parameters and report fields. This data can be processed using the report variables and their expressions. There are situations when a complex functionality cannot be achieved easily using report expressions or variables. Examples of this may be complex String manipulations, building of Maps or Lists of objects in memory or manipulations of dates using 3rd party Java APIs. For such situations, JasperReports provides us with a simple and powerful means of doing this with **Scriptlets**.

Scriptlets are sequences of Java code that are executed every time a report event occurs. Values of report variables can be affected through scriptlets.

Scriptlet Declaration

We can declare a scriptlet in two ways:

- Using **<scriptlet>** element. This element has *name* attribute and *class* attribute. The *class* attribute should specify the name of the class, which extends *JRAbstractScriptlet* class. The class must be available in the classpath at report filling time and must have an empty constructor, so that the engine can instantiate it on the fly.
- Using the attribute **scriptletClass** of the element **<jasperReport>**, in the report template (JRXML). By setting this attribute with fully qualified name of scriptlet (including the entire package name), we indicate that we want to use a scriptlet. The scriptlet instance created with this attribute, acts like the first scriptlet in the list of scriptlets and has the predefined name REPORT.

Scriptlet class

A scriptlet is a java class which must extend either of the following classes:

- **net.sf.jasperreports.engine.JRAbstractScriptlet**: This class contains a number of abstract methods that must be overridden in every implementation. These methods are called automatically by JasperReports at the appropriate moment. Developer must implement all the abstract methods.
- **net.sf.jasperreports.engine.JRDefaultScriptlet**: This class contains default empty implementations of every method in JRAbstractScriptlet. A developer is only required to implement those methods he/she needs for their project.

Following table lists the methods in the above class. These methods will be called by the report engine at the appropriate time, during report filling phase.

Method	Description
public void beforeReportInit()	Called before report initialization.
public void afterReportInit()	Called after report initialization.
public void beforePageInit()	Called before each page is initialized.
public void afterPageInit()	Called after each page is initialized.
public void beforeColumnInit()	Called before each column is initialized.
public void afterColumnInit()	Called after each column is initialized.

public void beforeGroupInit(String groupName)	Called before the group specified in the parameter is initialized.
public void afterGroupInit(String groupName)	Called after the group specified in the parameter is initialized.
public void beforeDetailEval()	Called before each record in the detail section of the report is evaluated.
public void afterDetailEval()	Called after each record in the detail section of the report is evaluated.

Any number of scriptlets can be specified per report. If no scriptlet is specified for a report, the engine still creates a single JRDefaultScriptlet instance and registers it with the built-in REPORT_SCRIPTLET parameter.

We can add any additional methods we need to our scriptlets. Reports can call these methods by using the built-in parameter REPORT_SCRIPTLET.

Global Scriptlets

We can associate scriptlets in another way to reports, which is by declaring the scriptlets globally. This makes the scriptlets apply to all reports being filled in the given JasperReports deployment. This is made easy by the fact that scriptlets can be added to JasperReports as extensions. The scriptlet extension point is represented by the *net.sf.jasperreports.engine.scriptlets.ScriptletFactory* interface. JasperReports will load all scriptlet factories available through extensions at runtime. Then, it will ask each one of them for the list of scriptlets instances that they want to apply to the current report that is being run. When asking for the list of scriptlet instances, the engine gives some context information that the factory could use in order to decide which scriptlets actually apply to the current report.

Report Governors

Governors are just an extension of global scriptlets that enable us to tackle a problem of report engine entering infinite loop at runtime, while generating reports. Invalid report templates cannot be detected at design time, because most of the time the conditions for entering the infinite loops depend on the actual data that is fed into the engine at runtime. Report Governors help deciding whether a certain report has entered an infinite loop and they can stop it, preventing resource exhaustion for the machine that runs the report.

JasperReports has two simple report governors that would stop a report execution based on a specified maximum number of pages or a specified timeout interval. They are:

1. **net.sf.jasperreports.governors.MaxPagesGovernor**: This is a global scriptlet that is looking for two configuration properties to decide if it applies or not to the report currently being run. The configuration properties are:
   - net.sf.jasperreports.governor.max.pages.enabled=[true|false]
   - net.sf.jasperreports.governor.max.pages=[integer]
2. **net.sf.jasperreports.governors.TimeoutGovernor**: This is also a global scriptlet that is looking for the following two configuration properties to decide if it applies or not: The configuration properties are:
   - net.sf.jasperreports.governor.timeout.enabled=[true|false]
   - net.sf.jasperreports.governor.timeout=[milliseconds]

The properties for both governors can be set globally, in the jasperreports.properties file, or at report level, as custom report properties. This is useful because different reports can have different estimated size or timeout limits and also because you might want turn on the governors for all reports, while turning it off for some, or vice-versa.

Example

Let's write a scriptlet class (**MyScriptlet**). The contents of file C:\tools\jasperreports-5.0.1\test\src\com\tutorialspoint\MyScriptlet.java are as follows:

```
package com.tutorialspoint;

import net.sf.jasperreports.engine.JRDefaultScriptlet;
import net.sf.jasperreports.engine.JRScriptletException;


public class MyScriptlet extends JRDefaultScriptlet
{
 public void afterReportInit() throws JRScriptletException
   {
      System.out.println("call afterReportInit()");
     // this.setVariableValue("AllCountries", sbuffer.toString());
      this.setVariableValue("someVar", new String("This variable value was modified by the 
scriptlet."));
   }

   public String hello() throws JRScriptletException
   {
      return "Hello! I'm the report's scriptlet object.";
   }

}
```

Details of the above scriptlet class are as below:

- In the *afterReportInit* method, we set a value to the variable **"someVar"** this.setVariableValue("someVar", new String("This variable value was modified by the scriptlet.")).
- At the end of the class an extra method called **hello** has been defined. This is an example of a method that can be added to the Scriptlet that actually returns a value, rather than setting a Variable.

Next we will add the scriptlet class reference in our existing report template (Chapter Report designs). The revised report template (jasper_report_template.jrxml) are as follows. Save it to C:\tools\jasperreports-5.0.1\test directory:

```
<?xml version="1.0"?>
<!DOCTYPE jasperReport PUBLIC
"//JasperReports//DTD Report Design//EN"
"http://jasperreports.sourceforge.net/dtds/jasperreport.dtd">

<jasperReport xmlns="http://jasperreports.sourceforge.net/jasperreports"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://jasperreports.sourceforge.net/jasperreports
http://jasperreports.sourceforge.net/xsd/jasperreport.xsd"
name="jasper_report_template" pageWidth="595"
pageHeight="842" columnWidth="515"
leftMargin="40" rightMargin="40" topMargin="50" bottomMargin="50"
scriptletClass="com.tutorialspoint.MyScriptlet">
   <style name="alternateStyle" fontName="Arial" forecolor="red">
      <conditionalStyle>
         <conditionExpression>
            <![CDATA[new Boolean($V{countNumber}.intValue() % 2 == 0)]]>
         </conditionExpression>
         <style forecolor="blue" isBold="true"/>
      </conditionalStyle>
   </style>
   <parameter name="ReportTitle" />
   <parameter name="Author" />
```

```
    <queryString>
     <![CDATA[]]>
    </queryString>

    <field name="country" >
       <fieldDescription>
       <![CDATA[country]]>
       </fieldDescription>
    </field>

    <field name="name" >
       <fieldDescription>
       <![CDATA[name]]>
       </fieldDescription>
    </field>

    <variable name="countNumber"
       calculation="Count">
       <variableExpression><![CDATA[Boolean.TRUE]]>
       </variableExpression>
    </variable>

    <variable name="someVar" >
       <initialValueExpression>
         <![CDATA["This is the initial variable value."]]>
       </initialValueExpression>
    </variable>

    <title>
       <band height="100">
          <line>
             <reportElement x="0" y="0" width="515"
             height="1"/>
          </line>
          <textField isBlankWhenNull="true" bookmarkLevel="1">
             <reportElement x="0" y="10" width="515"
              height="30"/>
             <textElement textAlignment="Center">
               <font size="22"/>
             </textElement>
             <textFieldExpression >
               <![CDATA[$P{ReportTitle}]]>
             </textFieldExpression>
             <anchorNameExpression><![CDATA["Title"]]>
             </anchorNameExpression>
          </textField>
          <textField isBlankWhenNull="true">
             <reportElement  x="0" y="40" width="515" height="20"/>
             <textElement textAlignment="Center">
                  <font size="10"/>
             </textElement>
             <textFieldExpression >
             <![CDATA[$P{Author}]]>
             </textFieldExpression>
          </textField>
          <textField isBlankWhenNull="true">
              <reportElement  x="0" y="50" width="515"
              height="30" forecolor="#993300"/>
              <textElement textAlignment="Center">
                   <font size="10"/>
              </textElement>
              <textFieldExpression >
              <![CDATA[$V{someVar}]]>
              </textFieldExpression>
          </textField>

       </band>
    </title>

    <columnHeader>
       <band height="23">
          <staticText>
             <reportElement mode="Opaque" x="0" y="3"
```

```
            width="535" height="15"
            backcolor="#70A9A9" />
            <box>
            <bottomPen lineWidth="1.0"
            lineColor="#CCCCCC" />
            </box>
            <textElement />
            <text><![CDATA[]]>
            </text>
         </staticText>
         <staticText>
            <reportElement x="414" y="3" width="121"
            height="15" />
            <textElement textAlignment="Center"
            verticalAlignment="Middle">
                <font isBold="true" />
            </textElement>
            <text><![CDATA[Country]]></text>
         </staticText>
         <staticText>
            <reportElement x="0" y="3" width="136"
            height="15" />
            <textElement textAlignment="Center"
            verticalAlignment="Middle">
               <font isBold="true" />
            </textElement>
            <text><![CDATA[Name]]></text>
         </staticText>
      </band>
   </columnHeader>

   <detail>
      <band height="16">
         <staticText>
            <reportElement mode="Opaque" x="0" y="0"
            width="535" height="14"
            backcolor="#E5ECF9" />
            <box>
               <bottomPen lineWidth="0.25"
               lineColor="#CCCCCC" />
            </box>
            <textElement />
            <text><![CDATA[]]>
            </text>
         </staticText>
         <textField>
            <reportElement style="alternateStyle"
            x="414" y="0" width="121" height="15" />
            <textElement textAlignment="Center"
            verticalAlignment="Middle">
               <font size="9" />
            </textElement>
            <textFieldExpression >
            <![CDATA[$F{country}]]>
            </textFieldExpression>
         </textField>
         <textField>
            <reportElement x="0" y="0" width="136"
            height="15" />
            <textElement textAlignment="Center"
            verticalAlignment="Middle" />
            <textFieldExpression >
            <![CDATA[$F{name}]]>
            </textFieldExpression>
         </textField>
      </band>
   </detail>
   <summary>
         <band height="45">
            <textField isStretchWithOverflow="true">
               <reportElement x="0" y="10" width="515" height="15" />
               <textElement textAlignment="Center"/>
               <textFieldExpression >
```

```
                  <![CDATA["There are " +
                  String.valueOf($V{REPORT_COUNT}) +
                     " records on this report."]]>
               </textFieldExpression>

            </textField>
            <textField isStretchWithOverflow="true">
               <reportElement positionType="Float" x="0" y="30" width="515"
               height="15" forecolor="#993300" />
               <textElement textAlignment="Center">
               <font size="10"/>
               </textElement>
               <textFieldExpression >
                  <![CDATA[$P{REPORT_SCRIPTLET}.hello()]]>
               </textFieldExpression>
            </textField>
         </band>
   </summary>
</jasperReport>
```

The details of the revised report template is as below:

- We have referenced the MyScriptlet class in the attribute *scriptletClass* of <jasperReport> element.
- Scriptlets can only access, but not modify, report fields and parameters. However, scriptlets can modify report variable values. This can be accomplished by calling the setVariableValue() method. This method is defined in JRAbstractScriptlet class, which is always the parent class of any scriptlet. Here we have defined a variable *someVar*, which will be modified by the MyScriptlet to have the value *This value was modified by the scriptlet*.
- The above report template has a method call in the Summary band that illustrates how to write new methods (in scriptlets) and use them in the report template. (**$P{REPORT_SCRIPTLET}.hello**())

The java codes for report filling remain unchanged. The contents of the file **C:\tools\jasperreports-5.0.1\test\src\com\tutorialspoint\JasperReportFill.java** are as below.

```
package com.tutorialspoint;

import java.util.ArrayList;
import java.util.HashMap;
import java.util.Map;

import net.sf.jasperreports.engine.JRException;
import net.sf.jasperreports.engine.JasperFillManager;
import net.sf.jasperreports.engine.data.JRBeanCollectionDataSource;

public class JasperReportFill {
   @SuppressWarnings("unchecked")
   public static void main(String[] args) {
      String sourceFileName =
      "C://tools/jasperreports-5.0.1/test/jasper_report_template.jasper";

      DataBeanList DataBeanList = new DataBeanList();
      ArrayList<DataBean> dataList = DataBeanList.getDataBeanList();

      JRBeanCollectionDataSource beanColDataSource =
      new JRBeanCollectionDataSource(dataList);

      Map parameters = new HashMap();
      /**
       * Passing ReportTitle and Author as parameters
       */
      parameters.put("ReportTitle", "List of Contacts");
      parameters.put("Author", "Prepared By Manisha");

      try {
         JasperFillManager.fillReportToFile(
         sourceFileName, parameters, beanColDataSource);
      } catch (JRException e) {
         e.printStackTrace();
      }
   }
}
```

The contents of the POJO file **C:\tools\jasperreports-5.0.1\test\src\com\tutorialspoint\DataBean.java** are as below:

```
package com.tutorialspoint;

public class DataBean {
   private String name;
   private String country;

   public String getName() {
      return name;
   }

   public void setName(String name) {
      this.name = name;
   }

   public String getCountry() {
      return country;
   }

   public void setCountry(String country) {
      this.country = country;
   }
}
```

The contents of the file **C:\tools\jasperreports-5.0.1\test\src\com\tutorialspoint\DataBeanList.java** are as below:

```
package com.tutorialspoint;

import java.util.ArrayList;

public class DataBeanList {
   public ArrayList<DataBean> getDataBeanList() {
      ArrayList<DataBean> dataBeanList = new ArrayList<DataBean>();

      dataBeanList.add(produce("Manisha", "India"));
      dataBeanList.add(produce("Dennis Ritchie", "USA"));
      dataBeanList.add(produce("V.Anand", "India"));
      dataBeanList.add(produce("Shrinath", "California"));

      return dataBeanList;
   }

   /**
    * This method returns a DataBean object,
    * with name and country set in it.
    */
   private DataBean produce(String name, String country) {
      DataBean dataBean = new DataBean();
      dataBean.setName(name);
      dataBean.setCountry(country);
      return dataBean;
   }
}
```

Report Generation

We will compile and execute the above file using our regular ANT build process. The contents of the file build.xml (saved under directory C:\tools\jasperreports-5.0.1\test) are as below.

> *The import file - baseBuild.xml is picked from chapter [Environment Setup](#) and should be placed in the*

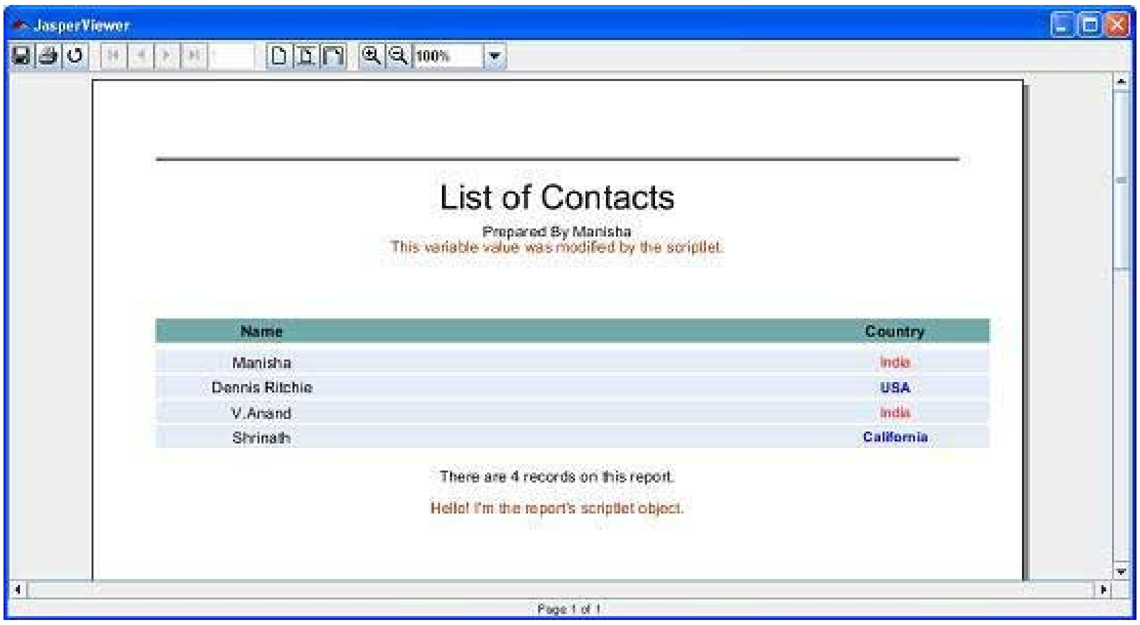
> *same directory as the build.xml.*

```
<?xml version="1.0" encoding="UTF-8"?>
<project name="JasperReportTest" default="viewFillReport" basedir=".">
   <import file="baseBuild.xml" />
   <target name="viewFillReport"
      depends="compile,compilereportdesing,run"
      description="Launches the report viewer to preview
      the report stored in the .JRprint file.">
      <java classname="net.sf.jasperreports.view.JasperViewer"
      fork="true">
         <arg value="-F${file.name}.JRprint" />
         <classpath ref />
      </java>
   </target>
   <target name="compilereportdesing"
      description="Compiles the JXML file and
      produces the .jasper file.">
      <taskdef name="jrc"
      classname="net.sf.jasperreports.ant.JRAntCompileTask">
         <classpath ref />
      </taskdef>
      <jrc destdir=".">
         <src>
         <fileset dir=".">
            <include name="*.jrxml" />
         </fileset>
         </src>
         <classpath ref />
      </jrc>
   </target>
</project>
```

Next, let's open command line window and go to the directory where build.xml is placed. Finally execute the command **ant -Dmain-class=com.tutorialspoint.JasperReportFill** (viewFullReport is the default target) as follows:

```
C:\tools\jasperreports-5.0.1\test>ant -Dmain-class=com.tutorialspoint.JasperReportFill
Buildfile: C:\tools\jasperreports-5.0.1\test\build.xml

clean-sample:
   [delete] Deleting directory C:\tools\jasperreports-5.0.1\test\classes
   [delete] Deleting: C:\tools\jasperreports-5.0.1\test\jasper_report_template.jasper
   [delete] Deleting: C:\tools\jasperreports-5.0.1\test\jasper_report_template.jrprint

compile:
    [mkdir] Created dir: C:\tools\jasperreports-5.0.1\test\classes
    [javac] C:\tools\jasperreports-5.0.1\test\baseBuild.xml:28:
    warning: 'includeantruntime' was not set, defaulting to bu
    [javac] Compiling 4 source files to C:\tools\jasperreports-5.0.1\test\classes

compilereportdesing:
      [jrc] Compiling 1 report design files.
      [jrc] log4j:WARN No appenders could be found for logger
      (net.sf.jasperreports.engine.xml.JRXmlDigesterFactory).
      [jrc] log4j:WARN Please initialize the log4j system properly.
      [jrc] log4j:WARN See http://logging.apache.org/log4j/1.2/faq.html#noconfig for more info.
      [jrc] File : C:\tools\jasperreports-5.0.1\test\jasper_report_template.jrxml ... OK.

run:
     [echo] Runnin class : com.tutorialspoint.JasperReportFill
     [java] log4j:WARN No appenders could be found for logger
     (net.sf.jasperreports.extensions.ExtensionsEnvironment).
     [java] log4j:WARN Please initialize the log4j system properly.
     [java] call afterReportInit()
     [java] call afterReportInit()

viewFillReport:
     [java] log4j:WARN No appenders could be found for logger
     (net.sf.jasperreports.extensions.ExtensionsEnvironment).
     [java] log4j:WARN Please initialize the log4j system properly.

BUILD SUCCESSFUL
Total time: 18 minutes 49 seconds
```

As a result of above compilation, a JasperViewer window opens up as in the screen below:



List of Contacts

Prepared By Manisha
This variable value was modified by the scriptlet.

Name	Country
Manisha	India
Dennis Ritchie	USA
V.Anand	India
Shrinath	California

There are 4 records on this report.

Hello! I'm the report's scriptlet object.

Here we see that in the title section a message *This variable value was modified by the scriptlet.* is displayed and at the bottom of the report a message *Hello! I'm the report's scriptlet object.* from the *hello* method of the MyScriptlet class is displayed.